**Vulnerability Assessment Report of**

**Q-Stress Thick Client Application – Failed Cases**

November 2022

**About L&T Technology Services:**

L&T Technology Services Limited (LTTS) is a global leader in Engineering and R&D (ER&D) services. With 399 patents filed for 51 of the Global Top 100 ER&D spenders. Our innovations speak for itself – World’s 1st Autonomous Welding Robot, Solar ‘Connectivity’ Drone, and the Smartest Campus in the World, to name a few. LTTS expertise in engineering design, product development, smart manufacturing, and digitalization touches every area of our lives. With 49 Innovation and R&D design centres globally, we specialize in disruptive technology spaces such as 5G, Artificial Intelligence, Collaborative Robots, Digital Factory, and Autonomous Transport. **LTTS is a publicly listed subsidiary of Larsen & Toubro Limited, the $18 billion Indian conglomerate operating in over 30 countries.**

Document History:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Var. Rel. No.** | **Release Date** | **Prepared By.**  **Prepared. Dt.** | **Reviewed By**  **Reviewed Dt.** | **Approved By**  **Approval Dt.** | **Remark/Revision Details** |
| 1.0 | 5th Dec 2022 | Sunil M C | Atanu / Jagdish | Atanu Niyogi |  |
| 5th Dec 2022 |  |  |

Table of Contents

[1. Overview of the project 4](#_Toc121126868)

[3. Abbreviation 24](#_Toc121126869)

# Overview of the project

L&T Technology Services (LTTS) has conducted Vulnerability Assessment & Penetration Testing (VAPT) of Q-Stress Thick Client Application. The purpose of the assessment is to evaluate the security posture of the Q-Stress Thick Client Application against common vulnerabilities with the primary reference being OWASP WEB Top 10 2021 & SANS TOP 25 Standard.

**Product Description**

Q-Stress is a diagnostic device that can display an ECG in real time, measure heart rate, perform ST analysis, and detect ventricular ectopic beats using wired or wireless acquisition modules. The device includes a touch screen interface, a keyboard/mouse interface, and a wired remote control unit. The device will keep a complete record of diagnostic quality test data that the user can use to generate and review stress test reports. The device can operate as a stand-alone workstation or connect to a database server via network, allowing for remote review. To obtain work lists and patient data, as well as to provide test result reports, the device can communicate with electronic record keeping systems.

**Objective of the security assessment:**

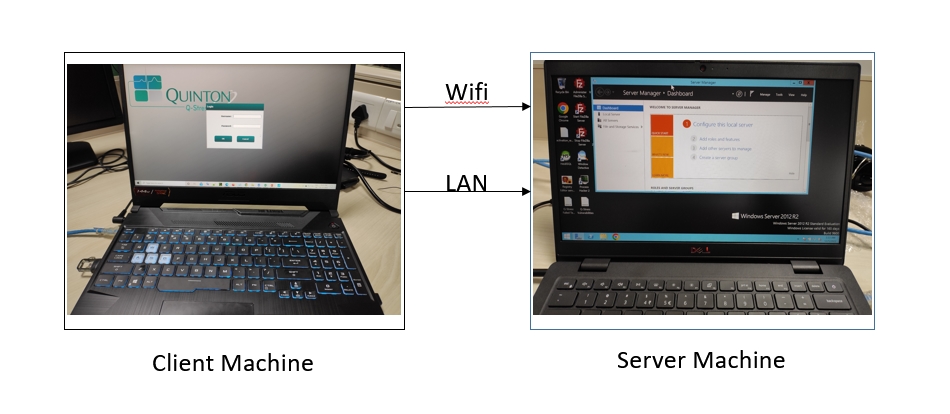
As a part of this engagement, a holistic approach was taken to conduct VAPT on Q-Stress Thick Client Application. During the engagement High, Medium and Informational severity issues were identified.

**Approach**

The Gray Box testing approach was taken to make sure the Product was assessed against vulnerabilities from all possible security perspectives:

* **Application profiling:** This includes enumerating the application's functionality and behaviour, understanding the application's core security mechanisms, identifying all the different entry points for user input, and determining which technologies are used on the Q-Stress application. This stage allows for the determination of the attack surface exposed by the application.
* **Authentication mechanism:** This includes testing the application's authentication-related functionality, such as login, password change.
* **Access controls:** This involves understanding the various access control requirements for the application and testing the implementation of access controls for defects leading to horizontal and vertical privilege escalation.
* **Input-based vulnerabilities:** This involves testing for input-based vulnerabilities like SQL injection and command injection. It involves fuzzing every request parameter with standard attack strings and manually investigating anomalous responses.
* **Sensitive data storage on files and registries:** During installation and execution, thick clients may write/modify sensitive files and registries. The sensitive data amassed by these applications usually contain username, passwords, database credentials, license details, cryptographic keys and configuration details like IP address, port, etc.
* **.DLL hijacking:** .DLL hijacking vulnerability involves attempt to hijack .DLL files that was loaded by a software installer from the directory where the installer is executed.
* **Network Analysis:** Intercept and thoroughly analyse network communication. Regardless of the protocol, use a proprietary tool to intercept and modify traffic.
* **Memory Analysis:** Test for sensitive information in live memory and dump memory of the application. Tampering with the memory to check for application behaviour.
* Manual Penetration Testing of using OWASP WEB Top 10 2021 & SANS Top 25 Standards.

**Test setup**



Some of the tools, which were used, are listed below:

|  |  |
| --- | --- |
| **Target Product** | Q-Stress Thick client Application |
| **Tools** | Strings, ProcMon, HxD, WinSpy++, Windows detective, WCFscan, Metasploit, DNSpy, Sysinternalsuite, TCPview, Wireshark, Burp Suite, Mitmreplay |

The Failed cases where attack vectors were carried out, but the system did not have the vulnerability is listed below

1. **Activation Abuse by Timestamp Tampering:**

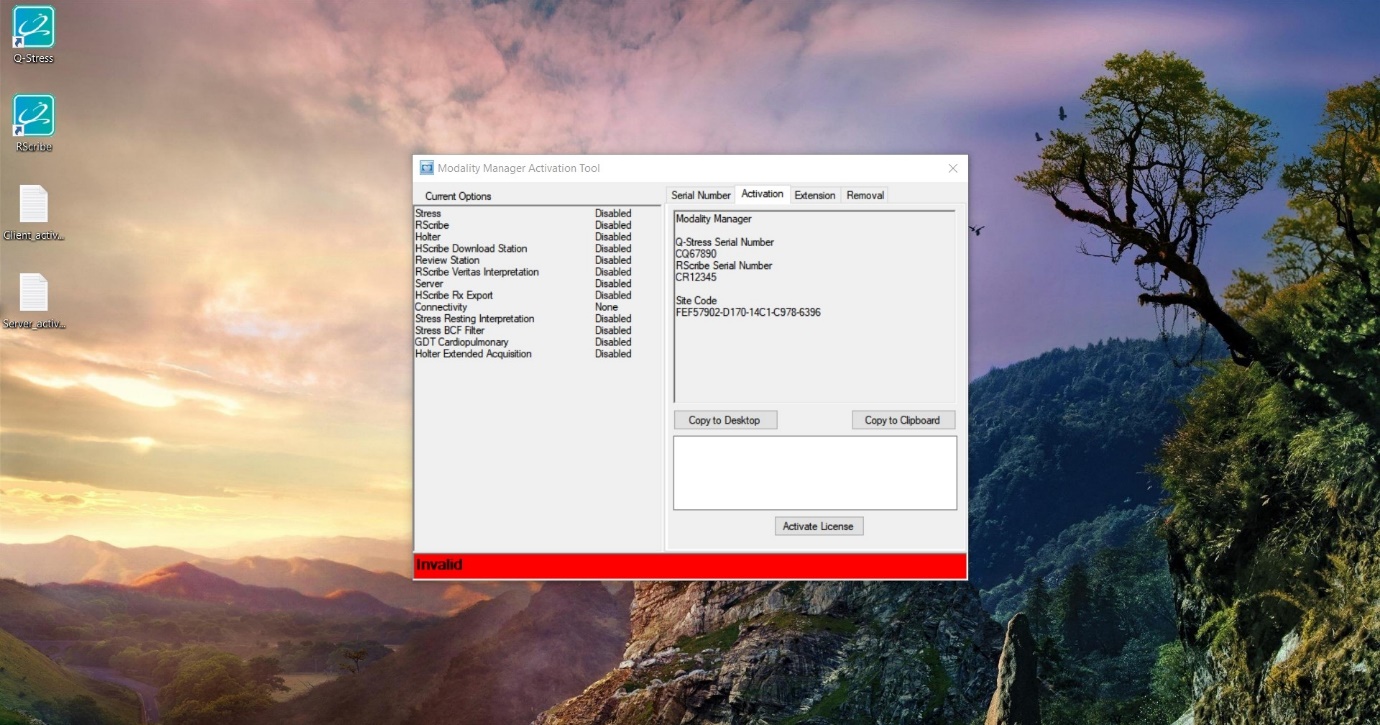
**Description:**

There is a 7-day free trial option for using the Q-Stress Application. After few days, change the system date and time to previous date and check whether the trial period has been extended.

**Expected Output:**

The expected output for this test case was whether the application accepts the given time stamp and work accordingly and thus, extend the free trial period, but the application reflects invalid message on Modality Manager Activation Tool.

**PoC (Proof of Concept):**

Fig 1.1: Invalid message pops up when changed the date & time of the system

1. **Creating a log File using Command Prompt:**

**Description:**

A log file is a computer-generated data file that contains valuable piece of information that is provided by the server. Almost all servers, services and applications provide some kind of logging. A log file records movements and actions that take place during the run time of a server or application.

**Expected Output:**

It is expected to generate a log file by running an application using CMD to check login log, services log and application’s activity log data.

**PoC (Proof of Concept):**

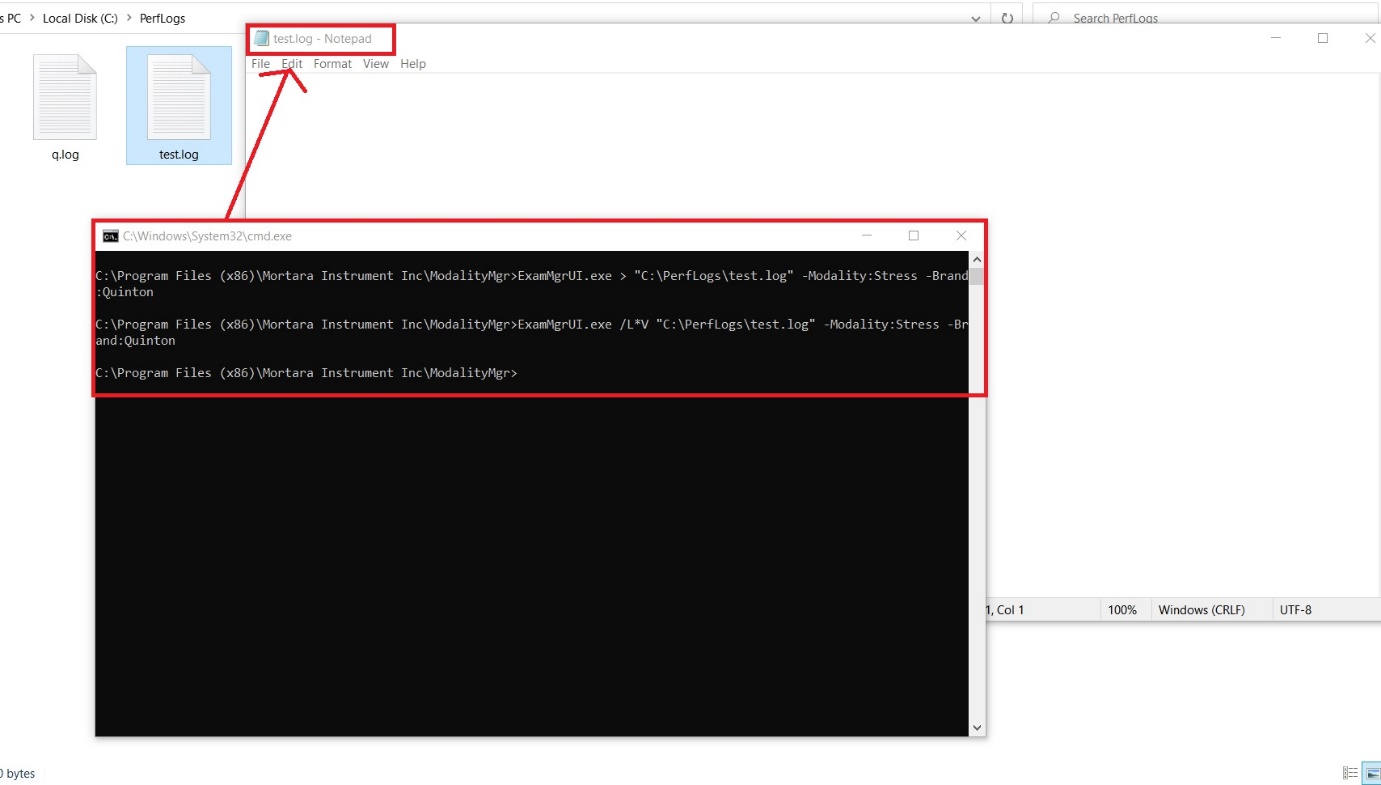


Fig 2.1: Log file generated using CMD but Application did not generate any log

1. **Creating a New Account with Existing Username:**

**Description:**

Creating a new user (low privilege) with the same username of admin. If successful in creating the user check for proper authentication during login. If no proper authentication is there then check if new user receives admin privilege in his account.

**Expected Output:**

The expected output was to create a new user with an existing username and sign in into the application with higher privilege account.

**PoC (Proof of Concept):**

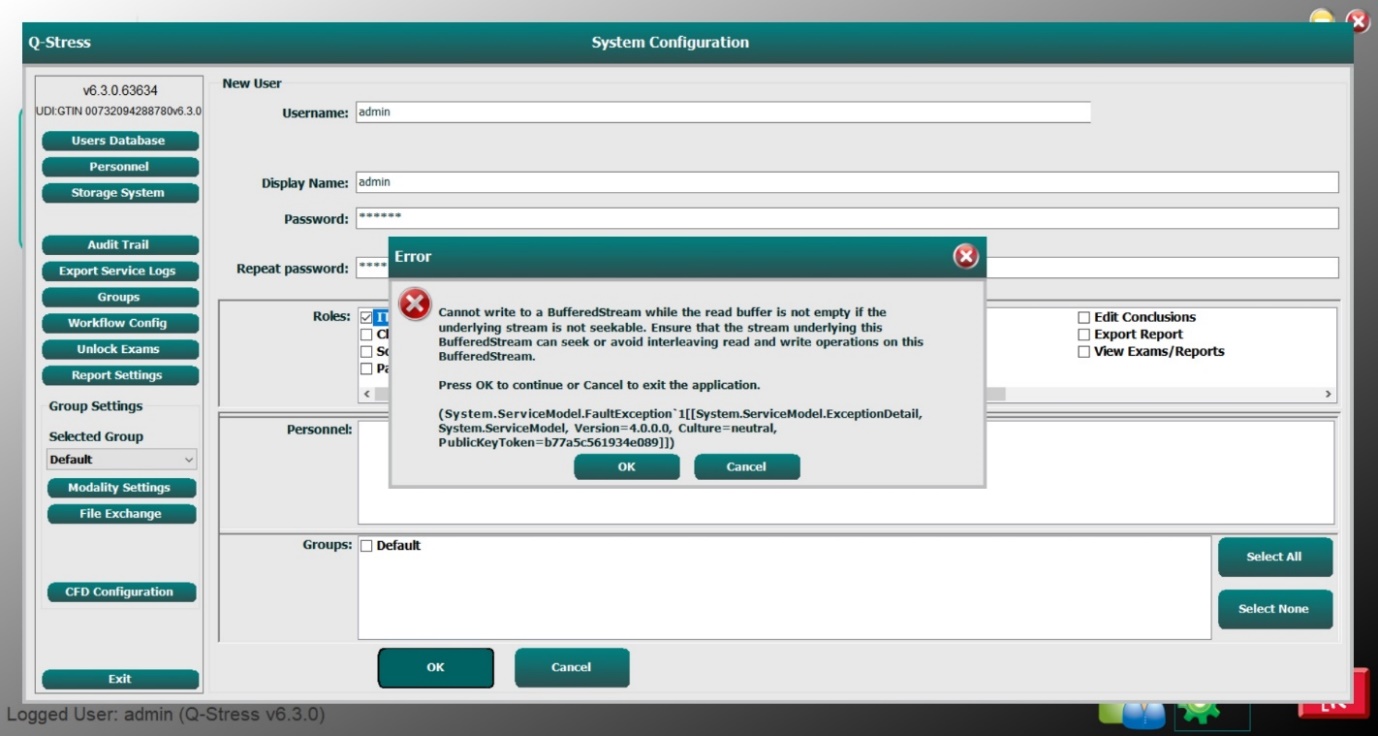


Fig 3.1: Same username not created

1. **Digital Signature Certificate:**

**Description:**

Digital Signature Certificate authenticates the identity of the signer digitally. It is basically an asymmetric cryptographic procedure that uses public and private keys generated by an algorithm, which makes it difficult to forge.

The application's digital signature was verified manually and with the **sigcheck64** tool.

**Expected Output:**

The expected output was that there should be no Digital Signature Certificate for the application.

**PoC (Proof of Concept):**

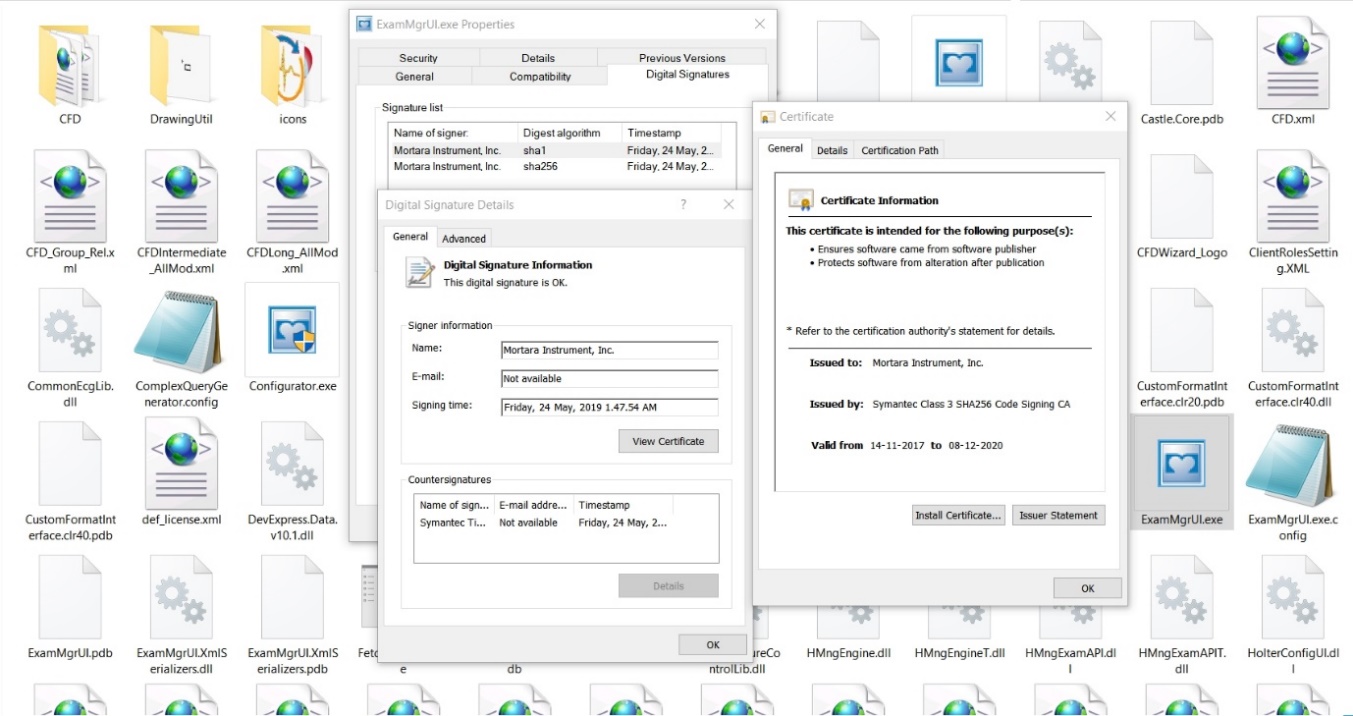


Fig 4.1: Digital Signature Certificate of the application present.

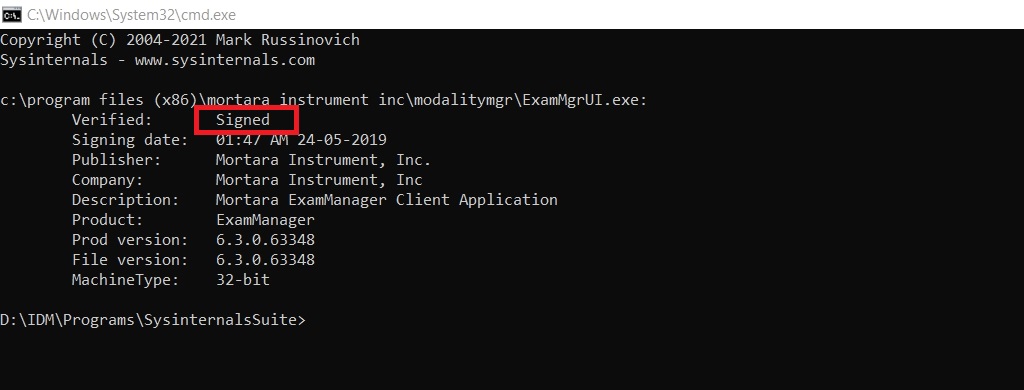


Fig 4.2: Digital Signature Certificate verified with sigcheck64 tool

1. **Hidden Admin UI Exposure:**

**Description:**

Logging into the programme as a client (low privilege) user and using the GUI client-side attack tool Windows Detective to grant the low privilege user all the admin capabilities available in the application.

**Expected Output:**

It was expected to use the Windows Detective tool to enable the user's database option and grant lower privilege user access to the user's database.

**PoC (Proof of Concept):**

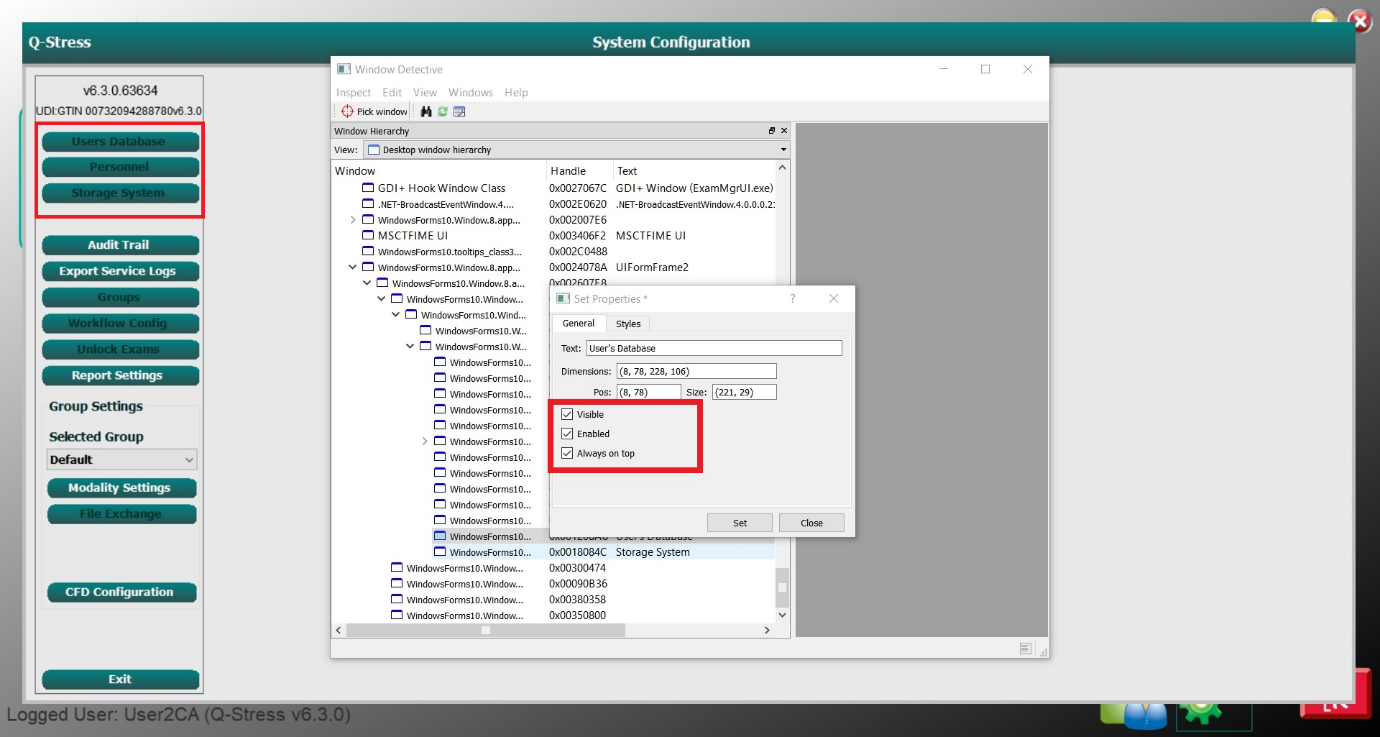


Fig 5.1: User’s Database option not enabled

1. **Sensitive Data in Registry (Server Installation and Client Installation):**

**Description:**

In the Registry file, the application stores data such as login credentials, user information and sensitive data (e.g. Admin/User credentials).

**Expected Output:**

The expected result was to locate any sensitive data or application information stored in Registry Files.

**PoC (Proof of Concept):**

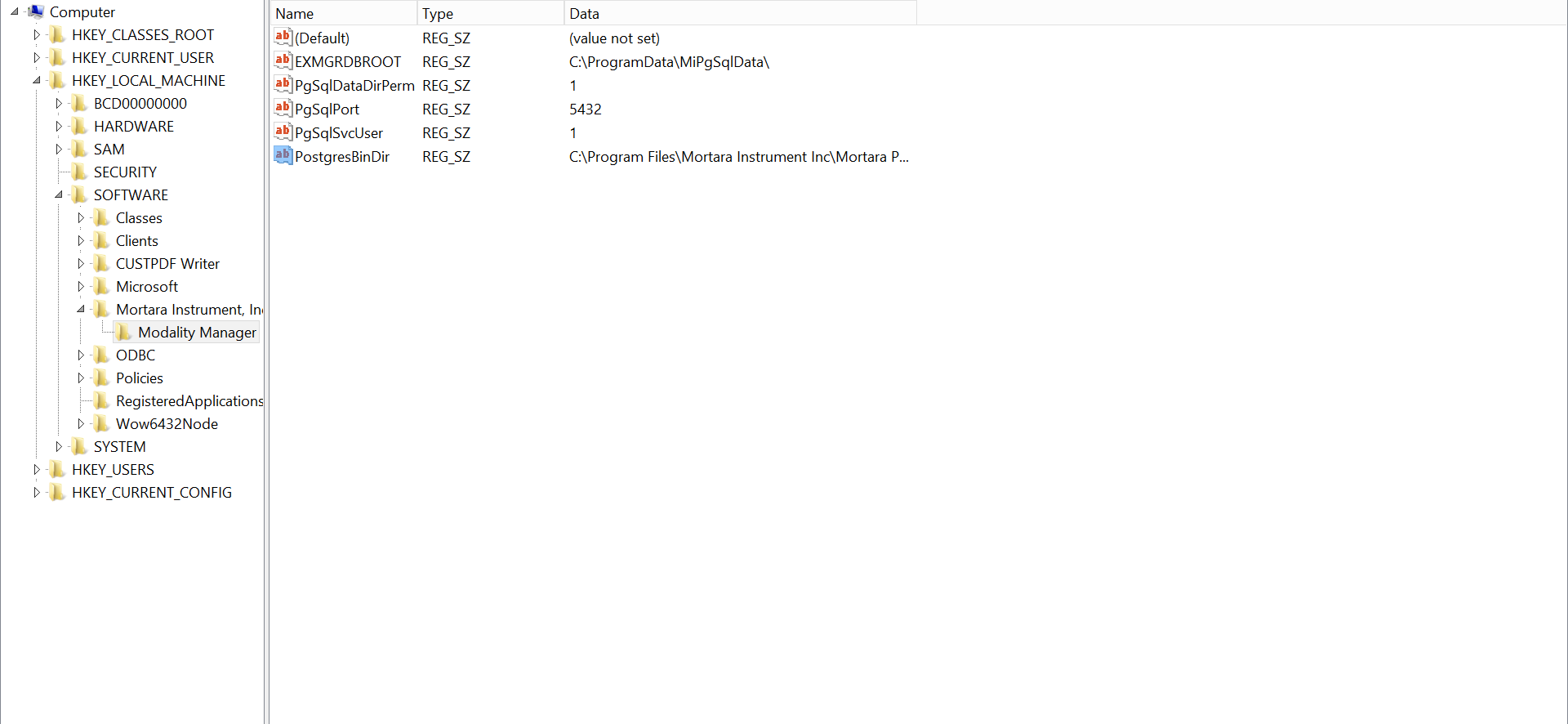


Fig 6.1: Application Registry Path (Server Installation)

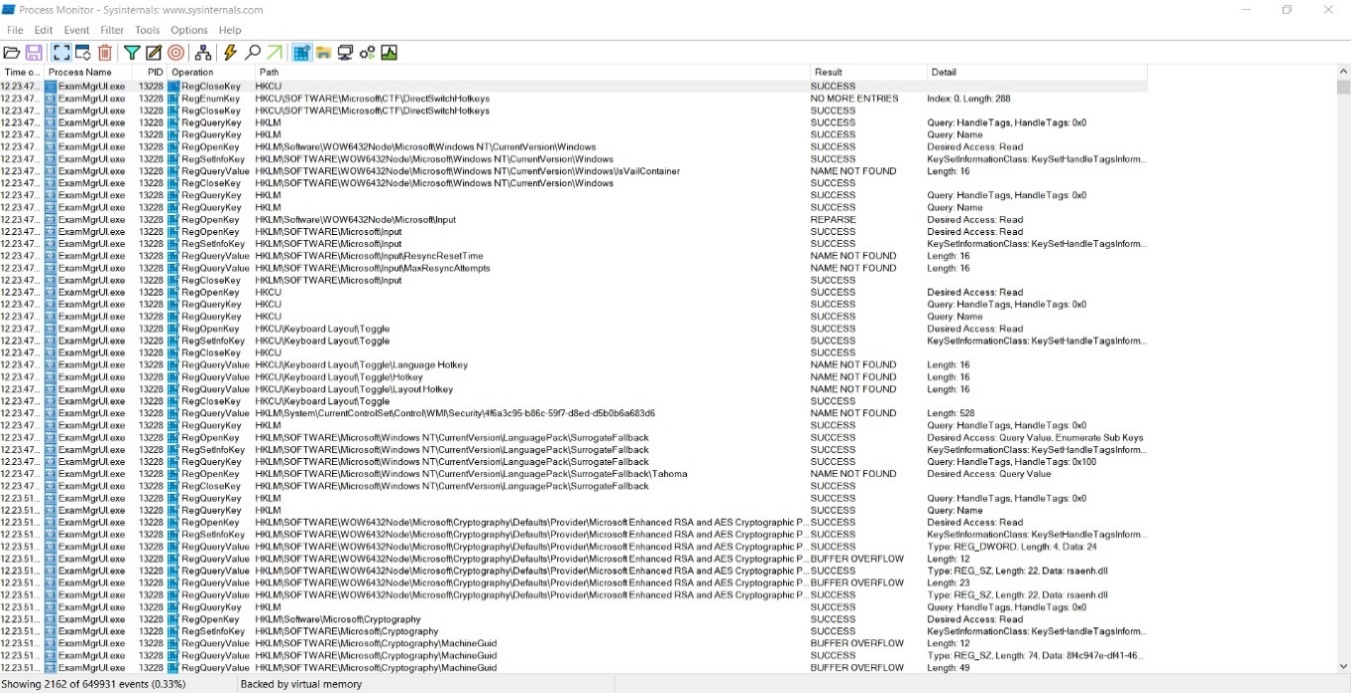


Fig 6.2: Application Registry Path through Procmon (Client Installation)

1. **Sensitive data in Live memory and Dump memory (Client installation and Server installation) :**

**Description:**

Observing live memory and analysing the dump memory for sensitive information in order to manipulate it and see if the changes are reflected in the application.

**Expected Output:**

Manipulated data in the live memory gets reflected in the application and sensitive information is found from the memory dump file.

**PoC (Proof of Concept):**

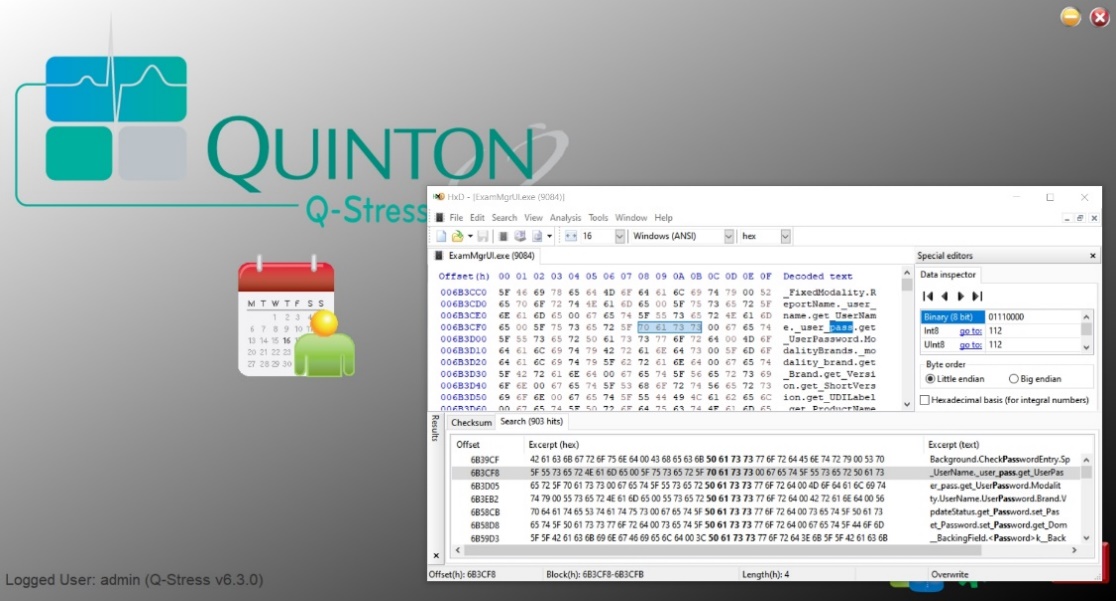


Fig 7.1: Live memory analysis using HxD tool.

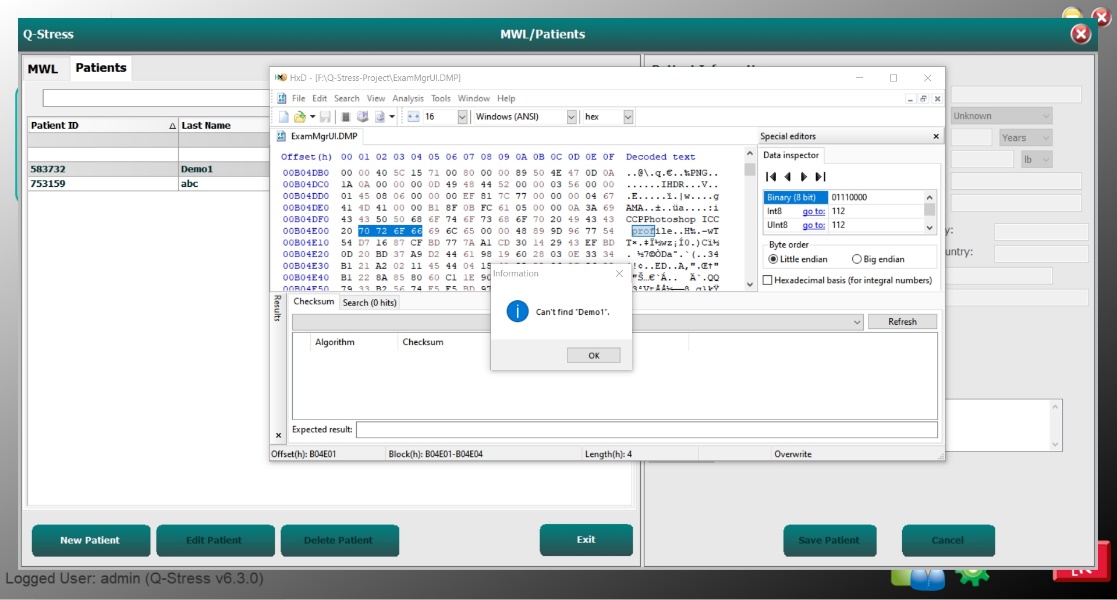


Fig 7.2: Dump memory analysis using HxD tool

1. **Hidden information and sensitive data in Export Service Log:**

**Description:**

Examining service logs for

1) Look for hidden URLs and see if they can be accessed via a web browser.

2) Sensitive information that could be used to further exploit the application

**Expected Output:**

Find sensitive information and use it to further exploit the application or access hidden URLs via a web browser.

**PoC (Proof of Concept):**

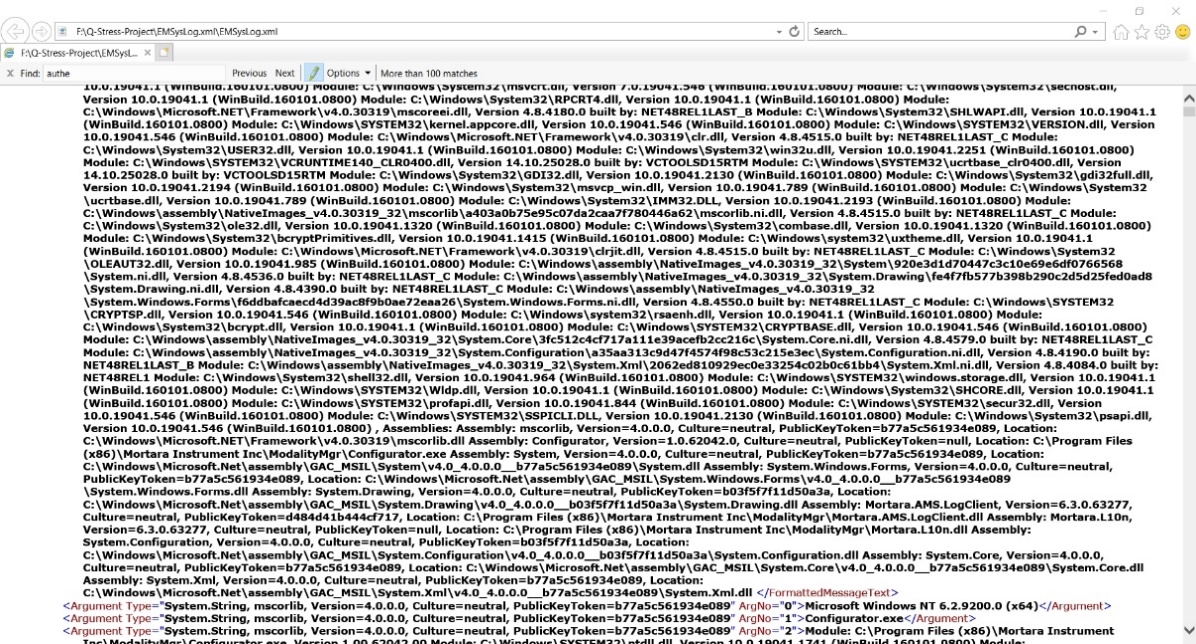
****

Fig 8.1: Service Log Analysis

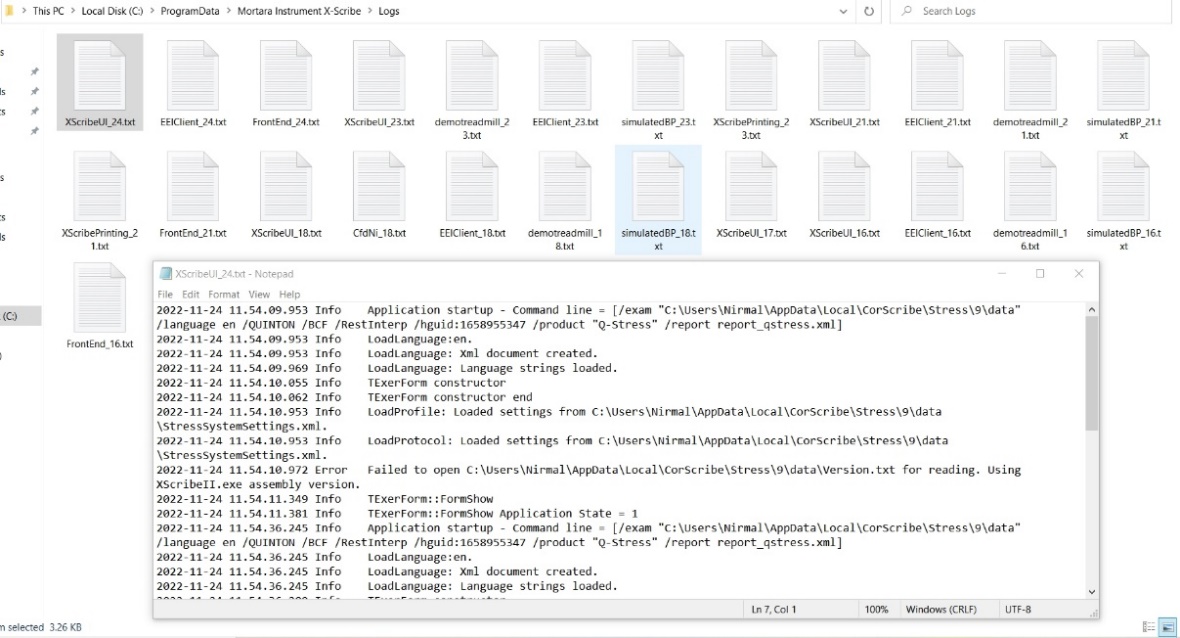


Fig 8.2: Log Analysis of Mortara Instrument

1. **Login Bypass using SQL Injection (GUI Level):**

**Description:**

SQL injection in the login field to bypass the authentication process The goal here was to bypass the application's authentication by providing some random expressions or commands that could pass with TRUE condition on the application's database and give us access to the application without having valid credentials.

The application's authentication by matching the TRUE condition on the database, granting us access to the application without valid credentials.

**Expected Output:**

The payload entered in the username and password fields was expected to be bypassed.

**PoC (Proof of Concept):**



Fig 9.1: SQLi to bypass Authentication

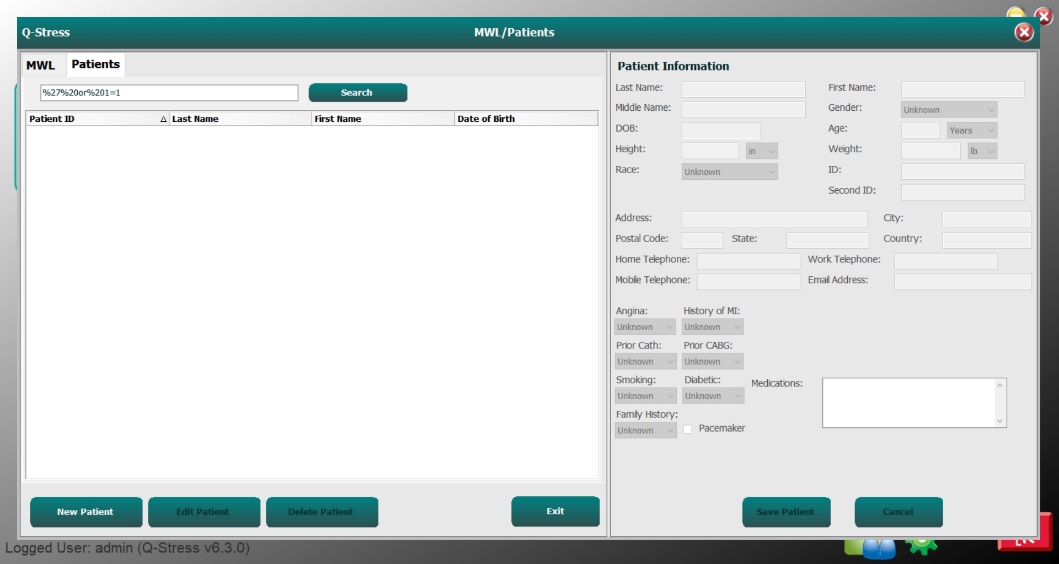


Fig 9.2: SQLi in Patient Searches.

1. **Insecure Folder Permission:**

**Description:**

To increase the impact of escalation of any vulnerability, we must have access to all files and folders in the application. The Q-Stress application's installation directory is not accessible to all user groups; it is only accessible to the NT authority group and the administration Group. Which, in this context, is preventing the existing vulnerability from being exacerbated.

**Expected Output:**

The expected result of this test case was to obtain EVERYONE as one of the folder access permissions, allowing us to copy/paste or manipulate the application's file and folder structure in order to amplify the impact of the existing vulnerabilities.

This test case failed because the folder permission provided was NT authority and administrator.

**PoC (Proof of Concept):**

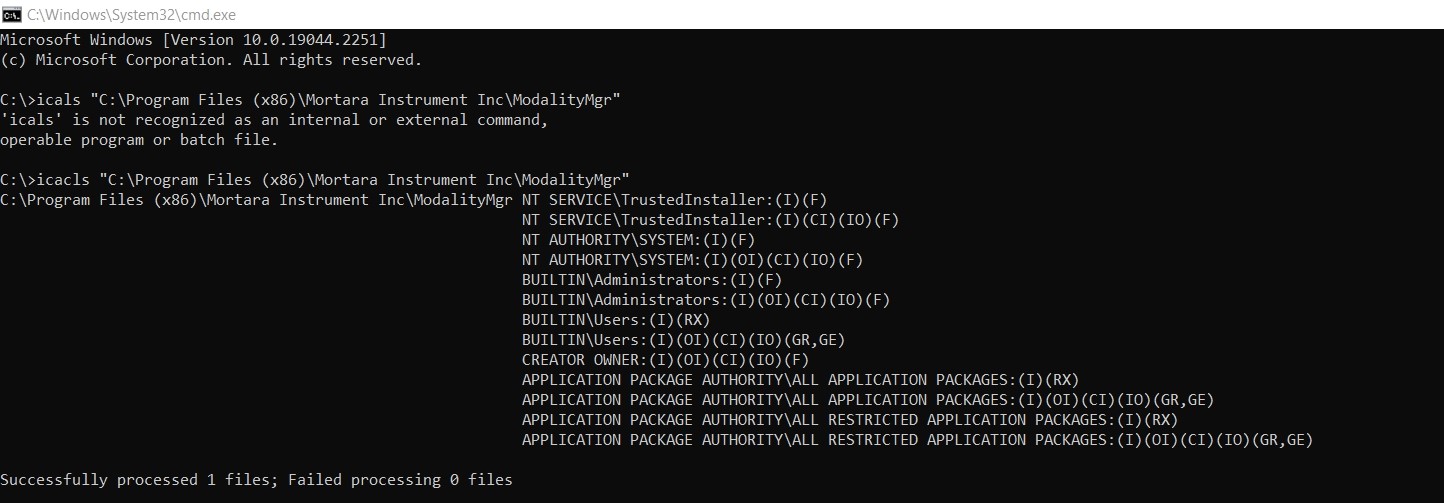


Fig 10.1: All the folder permissions in the application (there is no EVERYONE permission)

1. **Installation Folder Access to insert malicious file using Junction Point Creation:**

**Description:**

Creating a folder in F drive (Q-Stress) that does not require administrative privileges and linking it to the Q-Stress installation folder in C drive (MortalityMgr), then adding a malicious file to the F drive folder (Q-Stress) to determine whether administrative access is required to perform the action**.**

**Expected Output:**

It was anticipated that the malicious file would be successfully added to the F drive folder, which does not require admin privileges, and that the file would be added to the C drive folder (MortalityMgr) because the two folders are linked, thus bypassing the admin privilege requirement.

This test case failed because copying or moving files in the F drive requires administrator privileges.

**PoC (Proof of Concept):**

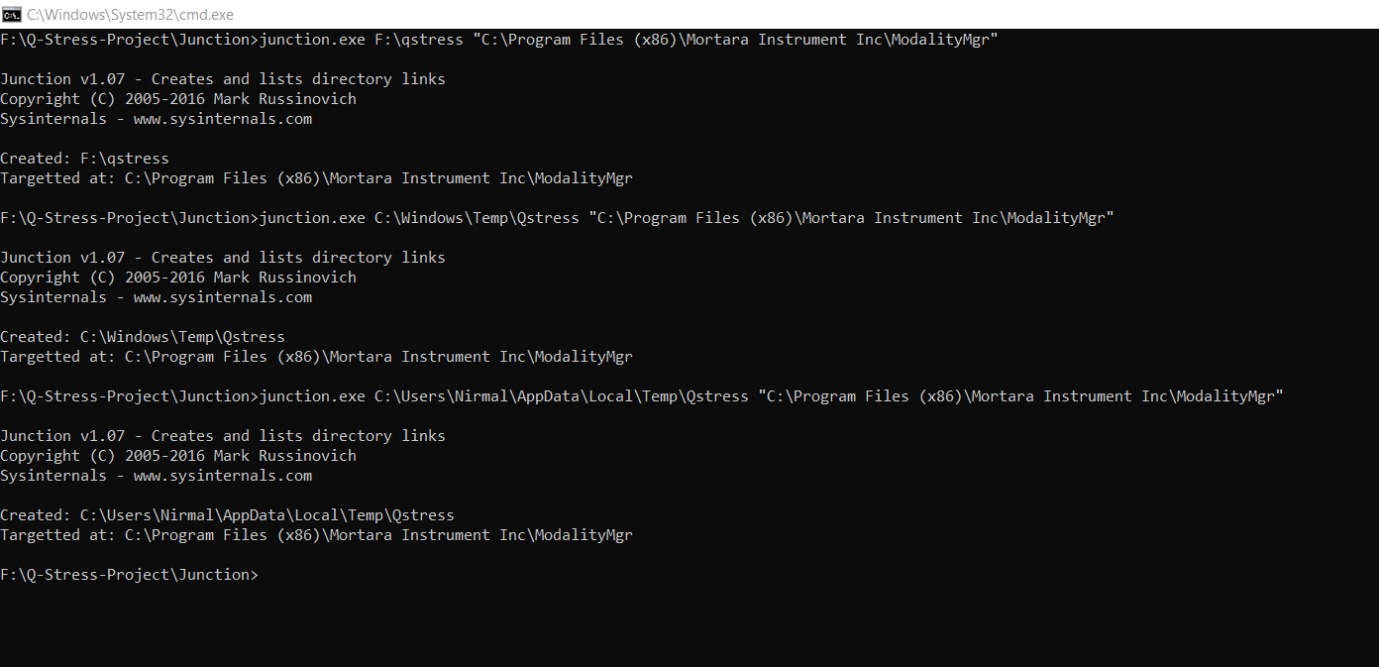


Fig 11.1: Creating a junction point using junction tool in CMD

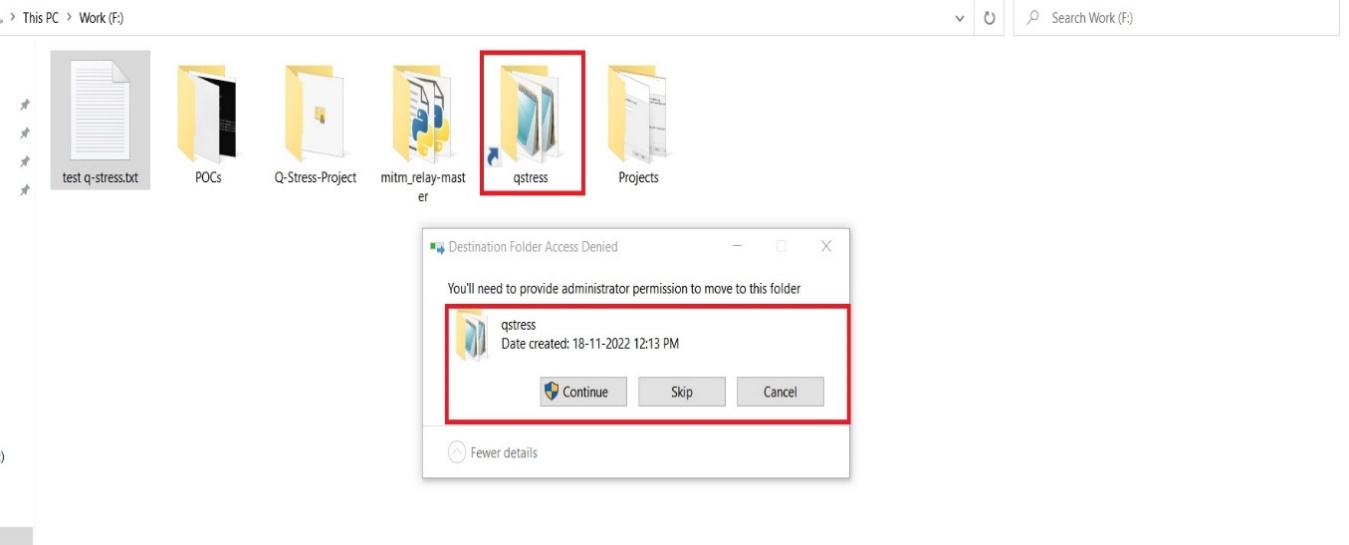


Fig 11.2: Creating a junction point with folder in F:drive (qstress)

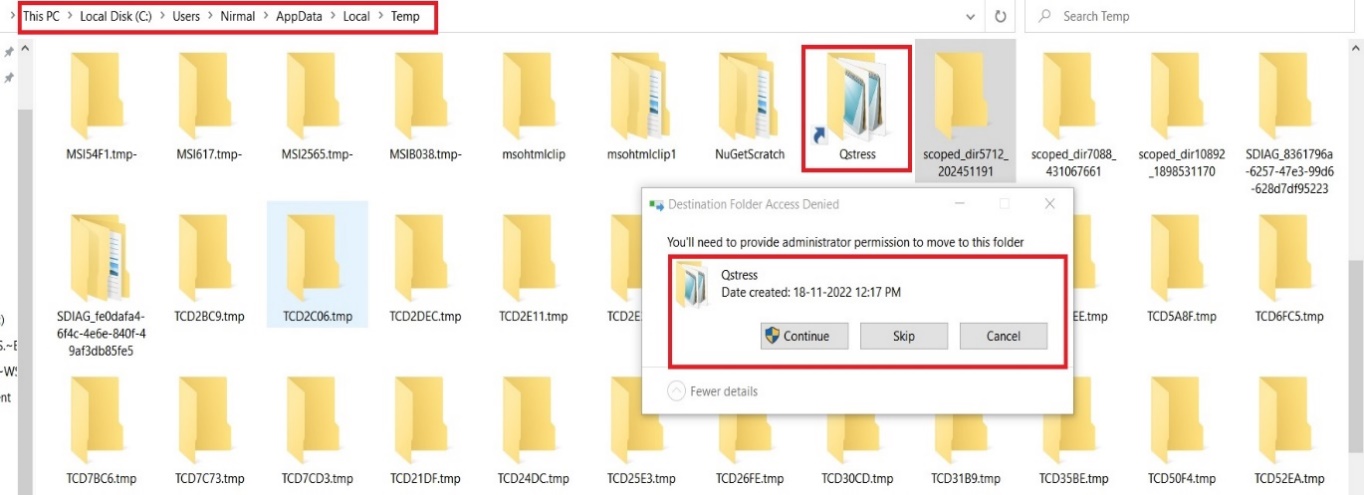


Fig 11.3: Creating a junction point with folder Program data Temp Folder (qstress).



Fig 11.4: Creating a junction point with folder in Windows Temp Folder (qstress).

1. **Clear Text Transmission in HTTP Traffic:**

**Description:**

Monitoring the network in Wireshark, a network monitoring tool, for HTTP requests in communication between client and server to ensure that no sensitive data is transmitted in clear text.

**Expected Output:**

It was expected to find an HTTP request between the client and the server, which could provide us with sensitive data in clear text format.

This test case failed because no HTTP requests were found in network traffic.

**PoC (Proof of Concept):**

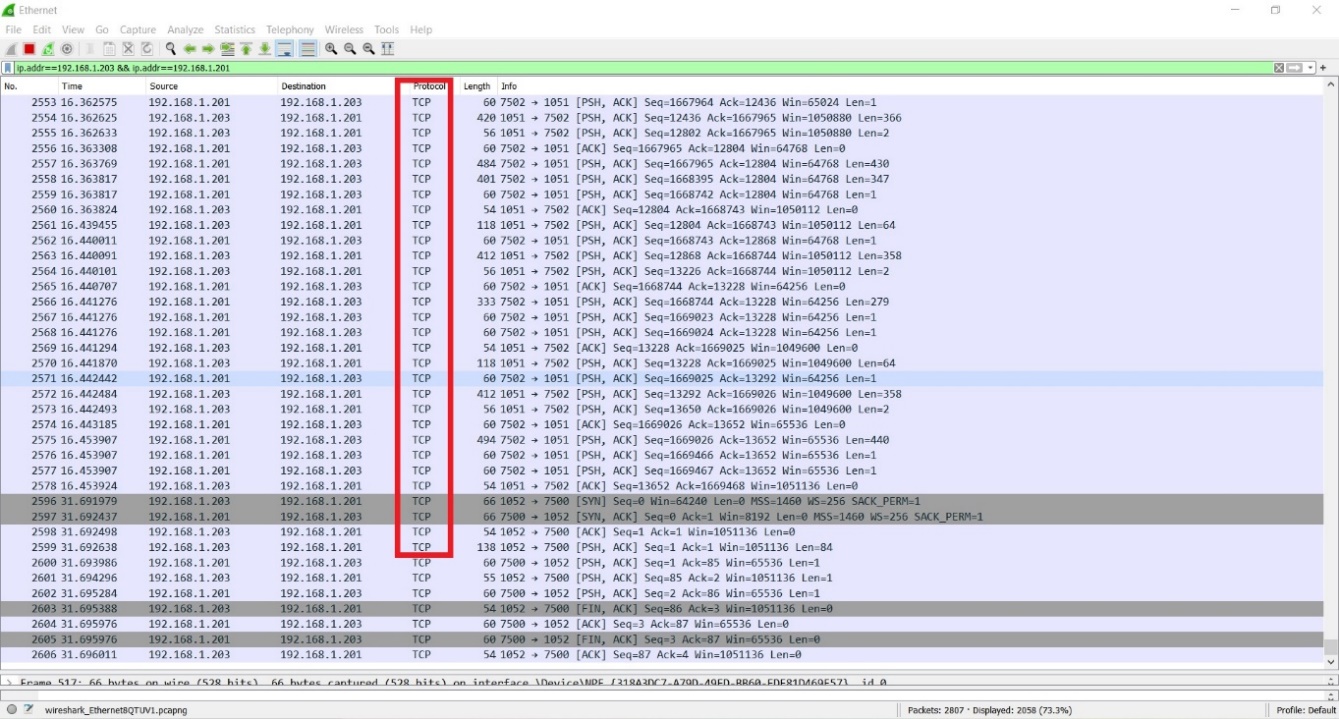


Fig 12.1: Monitoring the Network in Wireshark.

1. **TCP Replay Attack:**

**Description:**

In this test case, TCP traffic between the server and the client is intercepted. The network traffic packets on ports 7500 and 7502 are captured in a PCAP file, and these packets in the PCAP file are replayed to see if the application repeats the actions (creating users, changing roles, admin logging, etc.) specified in the PCAP file.

**Expected Output:**

The application was expected to repeat actions captured in network packets.

**PoC (Proof of Concept):**

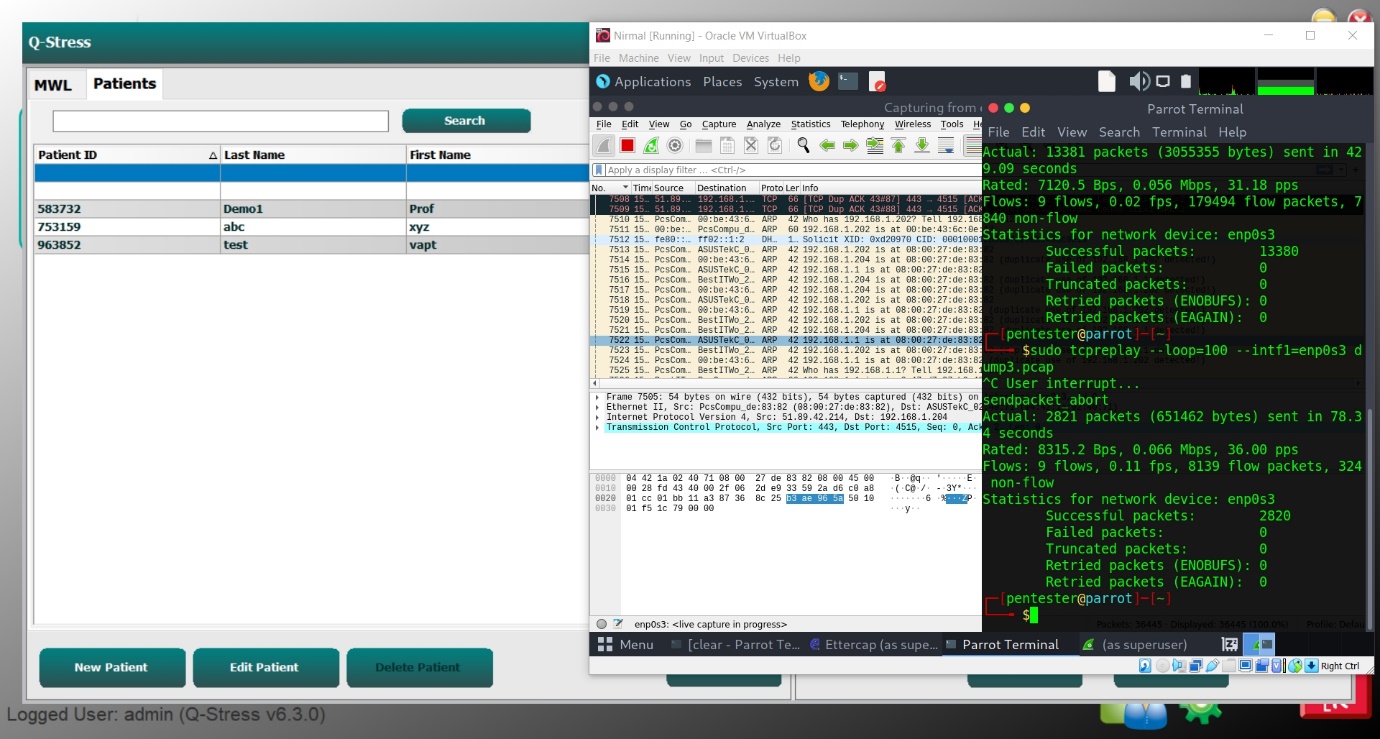


Fig 13.1: TCP replay on add patient

1. **Binary File Analysis using String Tool:**

**Description:**

Analysing all.exe files (ExamMgrUI.exe, ActivationTool.exe, cfdwizard.exe, Configurator.exe, CSDicomEcho.exe, DummyExamLoader.exe, FetchAllConfig.exe, MortaraCSPMgr.exe) to see if any sensitive information is stored in a binary file.

**Expected Output:**

The expected outcome of this test case was to obtain some sensitive information that could be used for further attacks or as a threat to the application.

**PoC (Proof of Concept):**

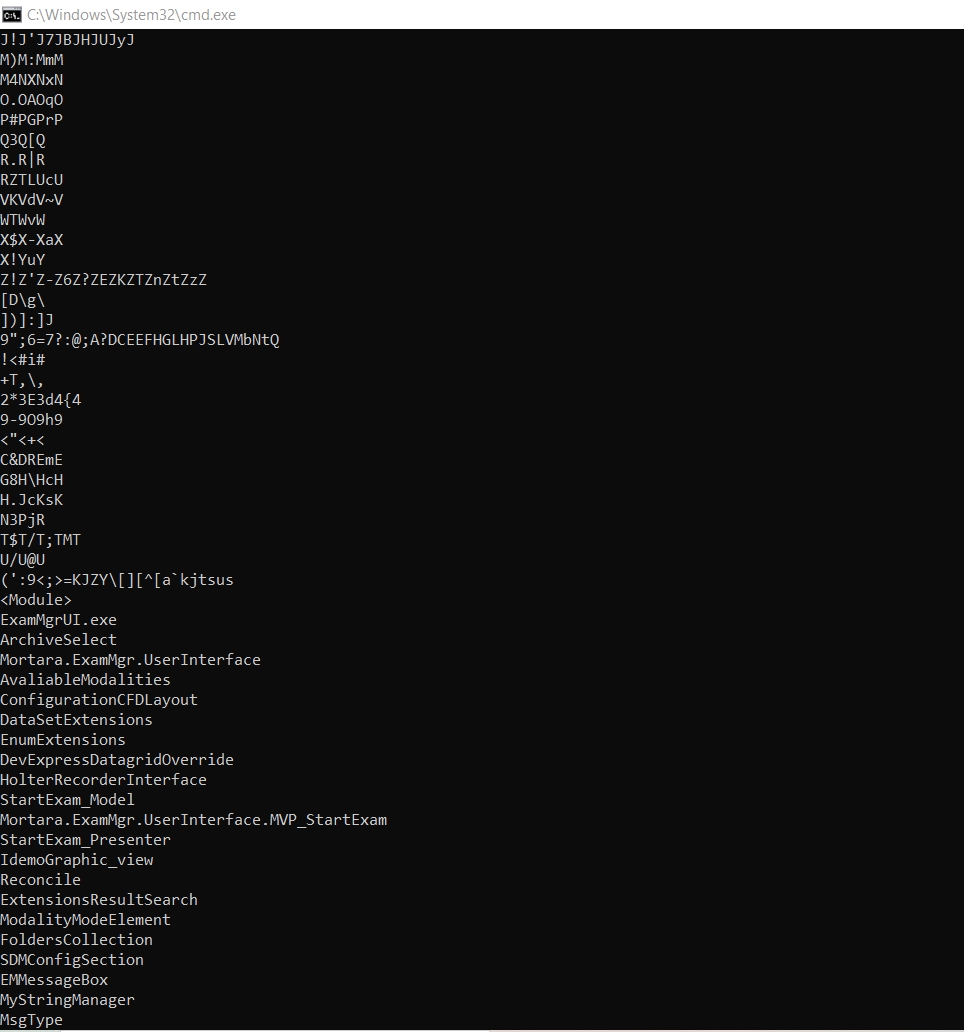


Fig 14.1: List of strings extracted using String tool.

1. **SQL Database Access from Client Machine:**

**Description:**

Database configuration information was discovered in the Configuration file. Using this configuration information, connect to the database with the HeidiSQL tool and perform Create, Update, and Delete operations on application data from the client machine.

**Expected Output:**

The client machine was expected to connect to the database using the information obtained from the configuration file.

The server refused the connection, so this test case failed.

**PoC (Proof of Concept):**

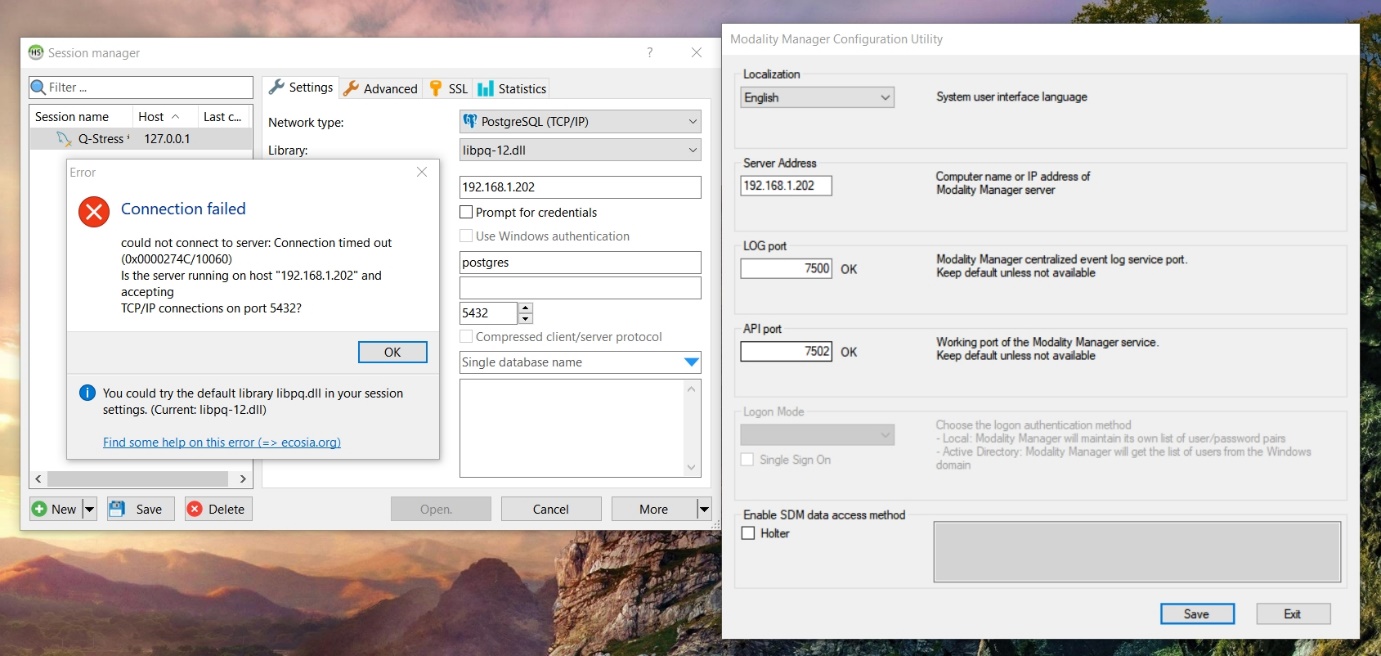


Fig 15.1: Unable to connect to server from HeldiSQL tool

1. **Accessing WCF.NET TCP Endpoint Configurations:**

**Description:**

The WCFSCAN tool was used to simply enumerate the transport and authentication settings of the.NET TCP WCF service's security configuration. WCFSCAN generates a very simple, generic service contract and attempts to connect to the supplied service endpoint URL, cycling through all possible security settings. Finally, the generic contract will not make a successful end-to-end call to the service; however, it will suffice for enumerating the security settings by interpreting the exceptions returned by the.NET framework.

**Expected Output:**

WCFSCAN enumerating a NET.TCP service that is wide open with no authentication, sending all of its messages in clear text across the network, and WCF Service Framework should return the following:

- The server accepted "None" mode.

- The server accepted the "Transport" mode.

- The server accepted the "Message" mode.

- If the server accepts the "TransportWithMessageCredential" mode

Then the server is open to the public with no authentication.

**PoC (Proof of Concept):**

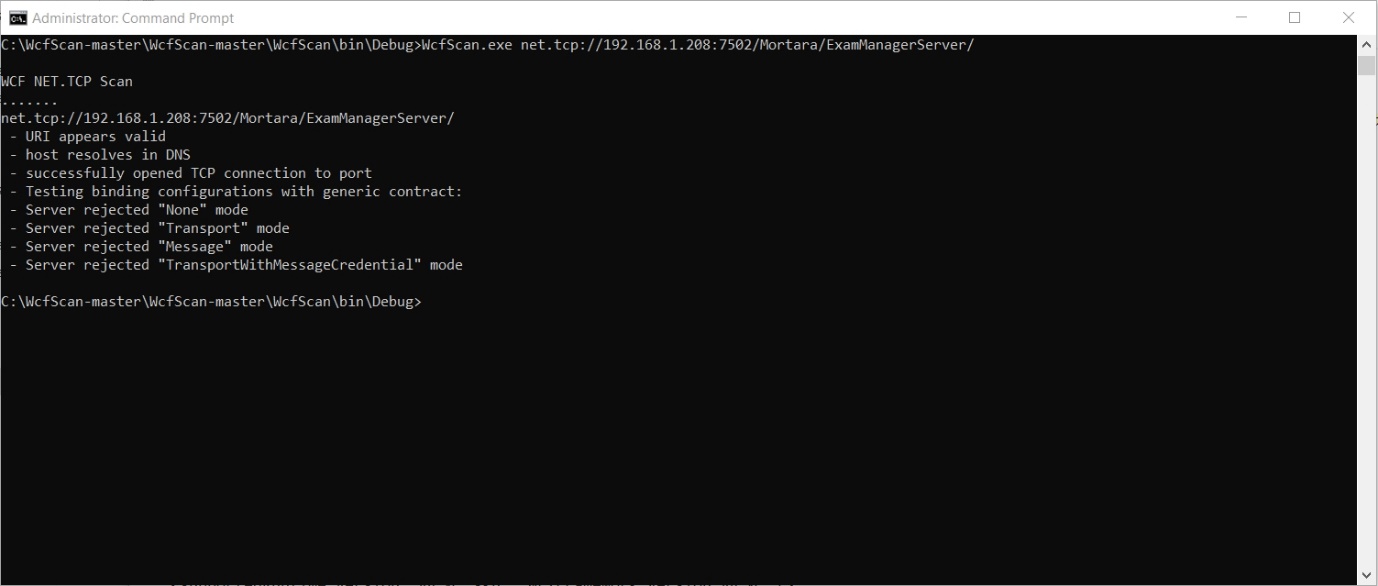


Fig 16.1: WCF.NET Binding server rejected modes

1. **Command Injection (GUI Level):**

**Description:**

The goal of command injection is to execute arbitrary commands on the host operating system via a vulnerable application.

**Expected Output:**

The expected output was to execute an OS command through any application input field.

**PoC (Proof of Concept):**

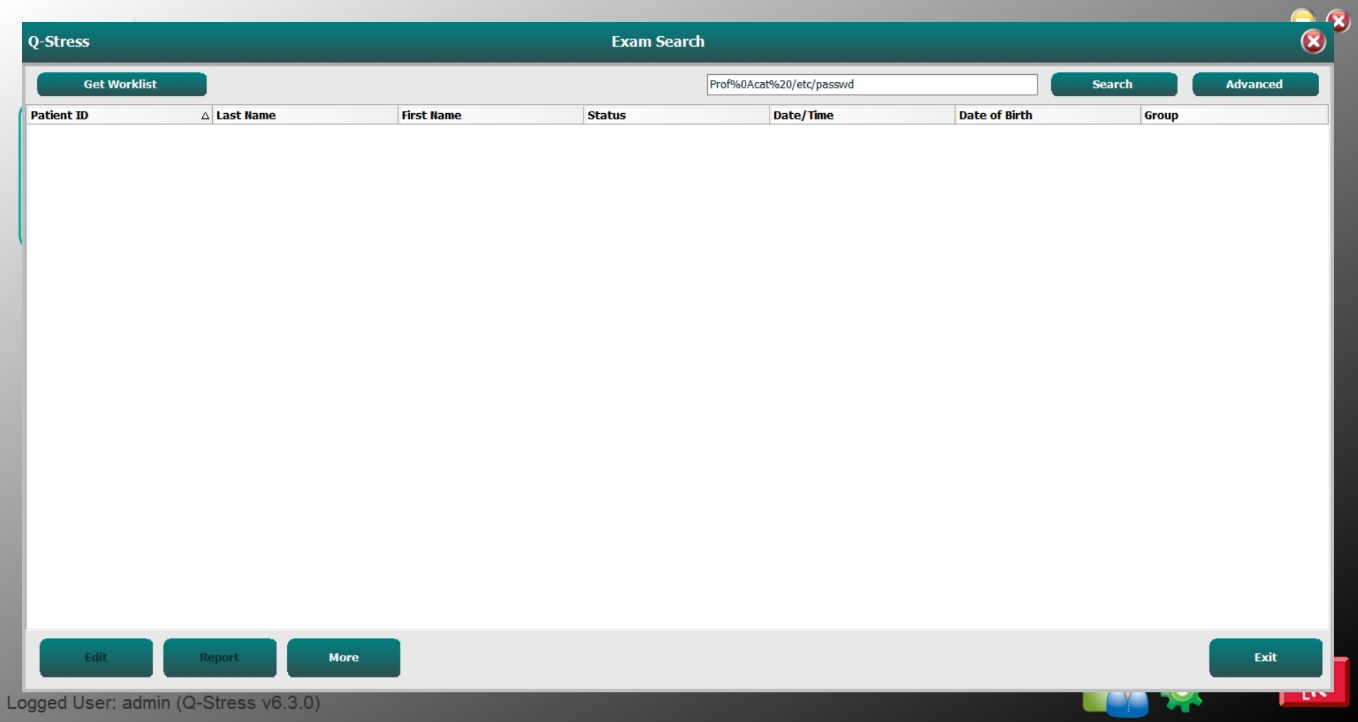


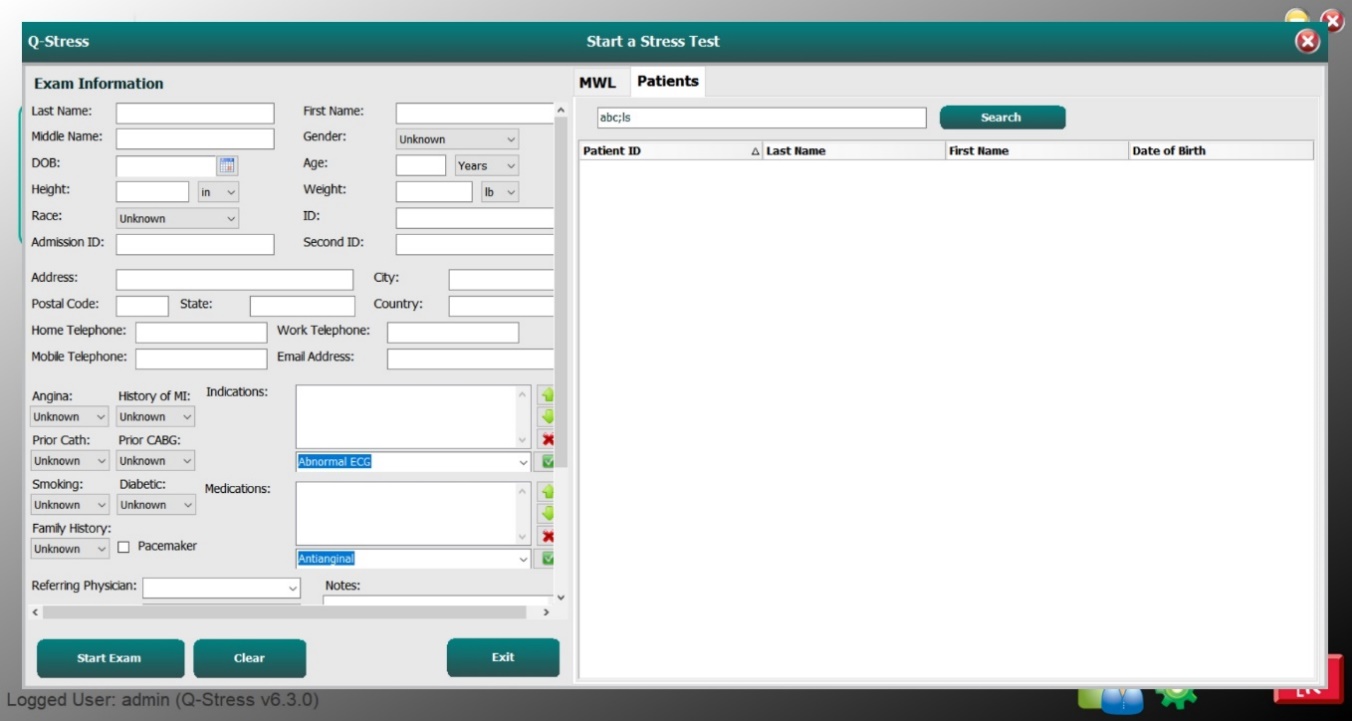
Fig 17.1: Command injection

Fig 17.2: Command injection

1. **Assembly Controls:**

**Description:**

Libraries and executables can be compiled with some additional security measures to protect against code exploitation such as Address Space Layout Randomization (ASLR), SafeSEH, Data Execution Prevention (DEP), Auntenticode/Strong Naming, Controlflowguard,HighentropyVA. All the security measures mentioned above were observed to be true.

**Expected Output:**

It was expected for the output to be False for various assembly control tests.

**PoC (Proof of Concept):**

Text

Description automatically generated

Fig .18.1: Output was True for assembly controls.

1. **Session Management:**

**Description:**

Improper Session management issue occurs when developers are not able to implement the session management best practices and if the session management is not properly done by the developers then an attacker may take the advantage of it and can compromise the user session which results in ATO(account take over).

**Expected Output:**

It was expected from the application to keep the session alive and not expire after certain period of time.

**PoC (Proof of Concept):**

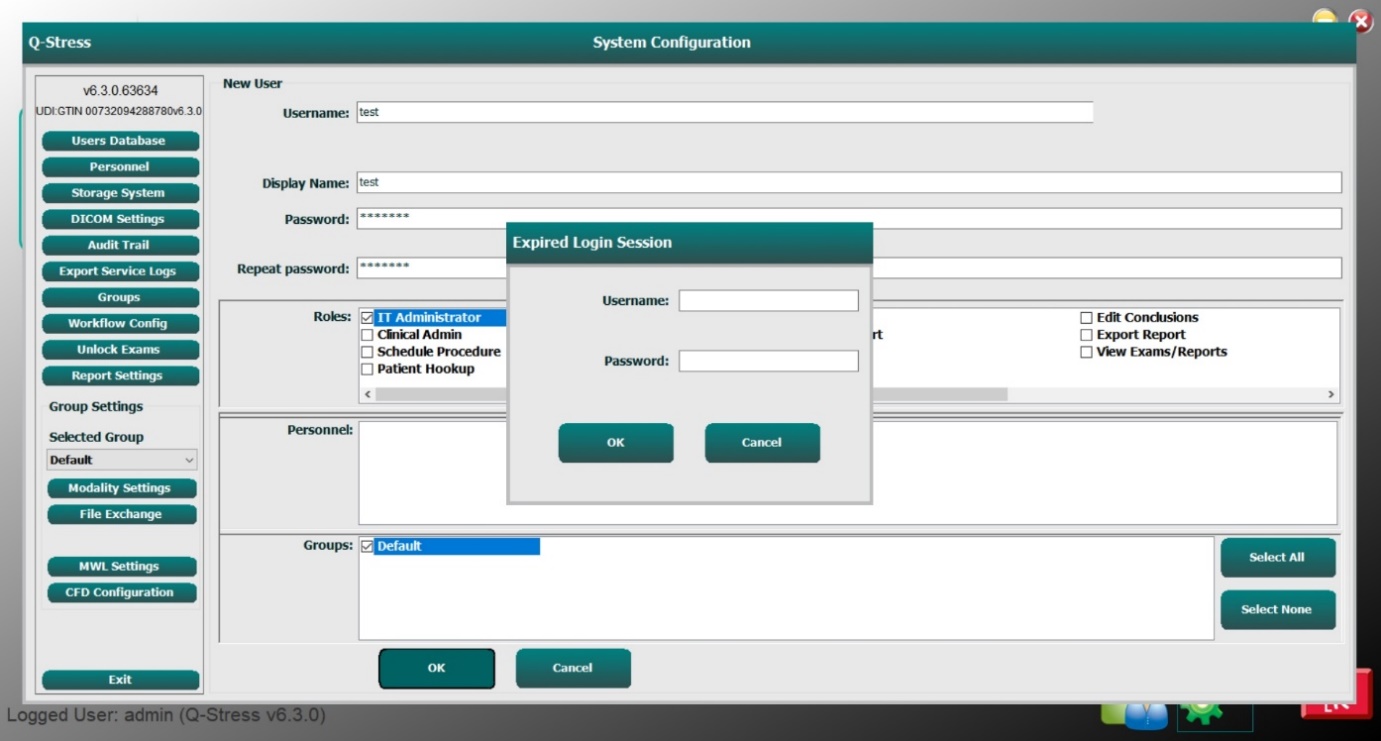
****

Fig .19.1: Login session expired

**Key Security Policies**

OWASP top listed vulnerabilities ware used as a reference framework. The following key security aspects were checked.

|  |  |
| --- | --- |
| **SR No** | **OWASP Web Top 10** |
| **1** | A01 Broken Access Control |
| **2** | A02 Cryptographic Failures |
| **3** | A03 Injection |
| **4** | A04 Insecure Design |
| **5** | A05 Security Misconfiguration |
| **6** | A06 Vulnerable and Outdated Components |
| **7** | A07 Identification and Authentication Failures |
| **8** | A08 Software and Data Integrity Failures |
| **9** | A09 Security Logging and Monitoring Failures |
| **10** | A10 Server-Side Request Forgery (SSRF) |

# Abbreviation

|  |  |
| --- | --- |
| APP | Application |
| HTML | HyperText Mark-up Language |
| HTTP(S) | Hypertext Transfer protocol (Secured) |
| Pg. | Page |
| TLS | Transport Layer Security |
| SSL | Secure Sockets Layer |
| IP | Internet Protocol |
| LTTS | Larsen & Toubro Technology Services |
| SOP | Same Origin Policy |
| OWASP | Open Web Application Security Project |
| VAPT | Vulnerability Assessment and Penetration testing |